# Java CM5

Olivier Marchetti

Laboratoire d'informatique de Paris 6 – Pôle SoC – Sorbonne Université

17 décembre 2021

# Plan



# Gestion des boutons à l'aide d'étiquettes

Les composants héritant de `AbstractButton` disposent notamment des méthodes :

```
String getActionCommand();
void setActionCommand(String etiquette);
```

$\Longrightarrow$ obtenir l'étiquette du composant émetteur

$\Longrightarrow$ définir l'étiquette du composant

Exemple : une fenêtre avec deux boutons avec usage exclusif.

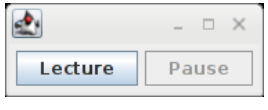
```
1  import javax.swing.*;
2  import static javax.swing.JFrame.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  class LectureEtPause
7      extends JFrame
8      implements ActionListener {
9
10     JButton lecture = new JButton("Lecture");
11     JButton pause = new JButton("Pause");
12
13     LectureEtPause() {
14         getContentPane().setLayout(new FlowLayout());
15         getContentPane().add(lecture);
16         getContentPane().add(pause);
17         lecture.addActionListener(this);
18         pause.addActionListener(this);
19         pause.setEnabled(false);
20     }                                                    ...
```

```
21
22     public void actionPerformed(ActionEvent e) {
23         if (e.getActionCommand() == "Lecture") {
24             lecture.setEnabled(false);
25             pause.setEnabled(true);
26         }
27         else if (e.getActionCommand() == "Pause") {
28             lecture.setEnabled(true);
29             pause.setEnabled(false);
30         }
31     }
32
33     public static void main(String args[]) {
34         LectureEtPause f = new LectureEtPause();
35         f.pack();
36         f.setDefaultCloseOperation(EXIT_ON_CLOSE);
37         f.setVisible(true);
38     }
39 }
```

Remarques :

1. Par défaut, l'étiquette d'un bouton est la chaîne de caractères visualisée ;

À l'exécution :




2. `setEnabled()` rend actif/inactif un composant.

# Faciliter la réalisation des interfaces utilisateurs – (1/2)

Soit une fenêtre qui réagirait à l'entrée et à la sortie du curseur de la souris dans la fenêtre.

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

class EcouteurFenetreMagique implements MouseListener {
    public void mouseClicked(MouseEvent e) {}

    public void mouseEntered(MouseEvent e) {
        System.out.println("Entrée");
    }

    public void mouseExited(MouseEvent e) {
        System.out.println("Sortie");
    }

    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}

    public static void main(String args[]) {
        JFrame f = new JFrame("Détection souris");
        f.addMouseListener(new EcouteurFenetreMagique());
        f.setBounds(400, 400, 500, 500);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Obligation d'implémenter toutes les méthodes de l'interface *MouseListener*

Il en résulte :

- des lignes de codes à taper/insérer (choisir un bon IDE), lignes pourtant ≪ inutiles ≫ ;
- une perte de lisibilité.

# Faciliter la réalisation des interfaces utilisateurs – classes XXXAdapter (2/2)

L'API JAVA prévoit :

$$\exists \text{ interface } \mathit{XXXListener} \Rightarrow \exists \text{ classe abstraite } \texttt{XXXAdapter}\ ^{1}$$

Classe adaptateur : ensemble des prototypes de l'interface associée.

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

class EcouteurFenetreMagique
    extends MouseAdapter {

    public void mouseEntered(MouseEvent e) {
        System.out.println("Entrée");
    }

    public void mouseExited(MouseEvent e) {
        System.out.println("Sortie");
    }

    public static void main(String args[]) {
        JFrame f = new JFrame("Détection souris");
        f.addMouseListener(new EcouteurFenetreMagique());
        f.setBounds(400, 400, 500, 500);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Rappel : par héritage, une classe dispose des méthodes de sa classe mère.

$\Longrightarrow$ redéfinir le strict nécessaire.

√ Gain de temps + lisibilité

À l'exécution :

```
[18:07][Prog pc666 :]$ java EcouteurFenetreMagique
Entrée
Sortie
Entrée
Sortie
```

[1] Attention à l'orthographe anglaise de ce mot... erreur de compilation pénible.

# Méthode MVC : principe

La réalisation d'une IU autorise plusieurs possibilités (*cf.* CM4 et ce cours), notamment pour le traitement des événements.

Cependant, une IU conséquente admet généralement un découpage naturel :

**Modèle**

(*i.e.* ensemble de classes gérant les données de l'application)

**Vue**

(*i.e.* ensemble de classes gérant la partie graphique)

**Contrôleur**

(*i.e.* ensemble de classes gérant les interactions avec l'application)

Ce découpage est ce que l'on appelle un patron de conception [1] .

Les difficultés pour le programmeur seront :

1. la définition d'un modèle rigoureux ;
2. l'identification des interactions souhaitables ;
3. la gestion des interactions avec son modèle.

[1] Aussi appelé *design pattern* en anglais.

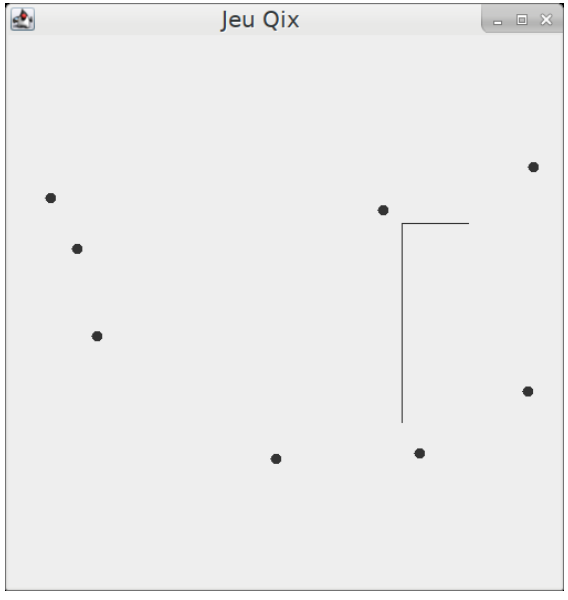
# Méthode MVC : un exemple

Le jeu du serpent consiste en :

- un serpent mobile de par sa tête et en perpétuel mouvement ;
- des cibles pour le serpent surgissant à des moments aléatoires (une cible mangée agrandira notre serpent) ;

et sous les contraintes que le serpent ne peut toucher :

- les parois de la fenêtre ;
- son propre corps.




**Modèle**

1. Le serpent ?
2. Les cibles ?

**Vue**

1. Une JFrame contenant un JPanel.
2. Dessin du serpent et des cibles.

**Contrôleur**

1. *Timer* pour animer le serpent.
2. Gestion des touches de direction du clavier.

# Les menus avec SWING

Pour réaliser un ensemble de menus, il faut adjoindre à la fenêtre un objet `JMenuBar` [1] :

```
JMenuBar objBarreMenu = new JMenuBar();
setJMenuBar(new JMenuBar());
```

Par la suite, il faut construire chacun des menus et les ajouter :

1. construire un menu de type `JMenu` [2] et l'ajouter :

```
JMenu objMenu = new JMenu();
objBarreMenu.add(objMenu);
```

2. construire une entrée de menu de type `JMenuItem` [3] et l'ajouter à un menu :

```
JMenuItem objEntreeMenu = new JMenuItem("Nom");
objMenu.add(new JMenuItem());
```

Une entrée de menu hérite de la classe `AbstractButton`.

3. Invalider éventuellement des menus ou des entrées :

```
objMenu.setEnabled(true/false);
objEntreeMenu.setEnabled(true/false);
```

[1] https://docs.oracle.com/javase/8/docs/api/javax/swing/JMenuBar.html
[2] https://docs.oracle.com/javase/8/docs/api/javax/swing/JMenu.html
[3] https://docs.oracle.com/javase/8/docs/api/javax/swing/JMenuItem.html

# Les menus avec SWING – exemple

```
import javax.swing.*;
import static javax.swing.JFrame.*;

class ExempleMenu extends JFrame {
    JMenuBar barre;
    JMenu fichier, ouvertRecemment, edition;
    JMenuItem ouvrir, imprimer, couper, copier, coller;

    ExempleMenu(String titre) {
        super(titre);
        // Création d'une barre de menus :
        barre = new JMenuBar();
        setJMenuBar(barre);
        // Création des menus :
        fichier = new JMenu("Fichier");
        fichier.setMnemonic('F');
        edition = new JMenu("Edition");
        barre.add(fichier);
        barre.add(edition);
        // Création des entrées des menus :
        ouvrir = new JMenuItem("Ouvrir");
        ouvertRecemment = new JMenu("Ouvert récemment");
        imprimer = new JMenuItem("Imprimer");
        imprimer.setEnabled(false);
        couper = new JMenuItem("Couper");
        copier = new JMenuItem("Copier");
        coller = new JMenuItem("Coller");
        fichier.add(ouvrir);
        fichier.add(ouvertRecemment);
        fichier.addSeparator();
        fichier.add(imprimer);
        edition.setEnabled(false);
        edition.add(couper);
        edition.add(copier);
        edition.add(coller);
    }
```

...

```
    public static void main(String args[]) {
        ExempleMenu f = new ExempleMenu("Fenetre avec menus");
        f.setSize(320, 125);
        f.setVisible(true);
        f.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

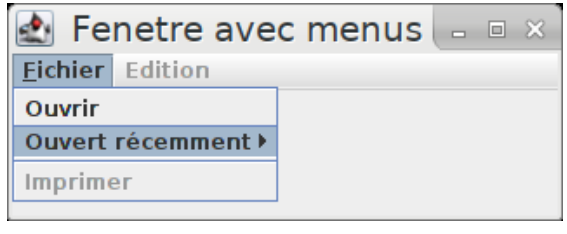
Remarques :

- ajout d'un « mnémonique » qui permettra d'ouvrir le menu avec ALT + F (cette lettre sera également soulignée si elle figure dans le nom du menu) ;

  `setMnemonic()` est héritée de la classe `AbstractButton`.

- possibilité d'ajouter des sous-menus ;
- possibilité d'ajouter un séparateur.

À l'exécution :



# Les menus avec SWING – programmation événementielle (1/3)

Pour rendre interactif un menu, on pourra implémenter au choix :

- *actionListener* associée aux événements de types `ActionEvent` ;

  $\Longrightarrow$ *cf.* cours précédent.

- *MouseListener* associée aux événements de types `MouseEvent` ;

  $\Longrightarrow$ *cf.* cours précédent.

- *ItemListener* [1] associée aux événements de type `ItemEvent` [2] ;

  $\Longrightarrow$ rédéfinir la méthode `itemStateChanged()`.

- *MenuListener* [3] associée aux évenements de type `MenuEvent` [4].

  $\Longrightarrow$ rédéfinir les méthodes `menuCanceled()`
  `menuDeselected()`
  `menuSelected()`

[1] https://docs.oracle.com/javase/8/docs/api/java/awt/event/ItemListener.html
[2] https://docs.oracle.com/javase/8/docs/api/java/awt/event/ItemEvent.html
[3] https://docs.oracle.com/javase/8/docs/api/java/awt/event/MenuListener.html
[4] https://docs.oracle.com/javase/8/docs/api/java/awt/event/MenuEvent.html

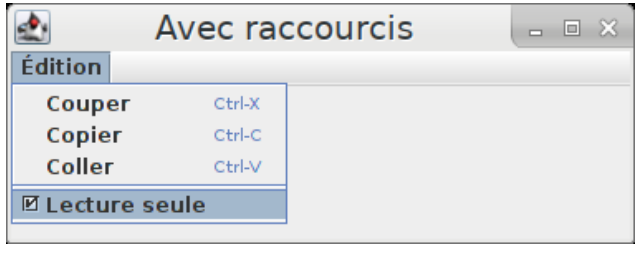
# Les menus avec SWING – programmation événementielle (2/3)

```
import javax.swing.*;
import static javax.swing.JFrame.*;
import java.awt.event.*;
import static java.awt.event.KeyEvent.*;
import static java.awt.event.InputEvent.*;

class ExempleMenuRaccourcis
    extends JFrame
    implements ActionListener, ItemListener {
    JMenuBar barre;
    JMenu edition;
    JMenuItem couper, copier, coller;
    JCheckBoxMenuItem lectureSeule;

    ExempleMenuRaccourcis(String titre) {
        super(titre);
        barre = new JMenuBar();
        setJMenuBar(barre);
        // Création des menus avec raccourcis :
        edition = new JMenu("Édition");
        barre.add(edition);
        // Création des entrées des menus :
        couper = new JMenuItem("Couper");
        KeyStroke combiTouches = KeyStroke.getKeyStroke(VK_X,
                                                        CTRL_MASK);
        couper.setAccelerator(combiTouches);
        couper.addActionListener(this);
        ... // idem avec copier et coller.
        lectureSeule = new JCheckBoxMenuItem("Lecture seule");
        lectureSeule.addItemListener(this);
        edition.add(couper);
        ...
        edition.addSeparator();
        edition.add(lectureSeule);
    }
                                                              ...
```

Remarques :

- insertion d'imports statiques pour disposer des noms cours des touches ;
- définition d'un objet de type `KeyStroke` permettant de définir des combinaisons de touches ;
- définition d'un raccourci clavier ou ≪ accélérateur ≫ ;
- ajout d'écouteurs de type
  - `ActionListener` ;
  - `ItemListener`.

Visuellement, à l'exécution :



# Les menus avec SWING – programmation événementielle (3/3)

```
    public void actionPerformed(ActionEvent e) {
        switch (e.getActionCommand()) {
        case "Couper":
            System.out.println("Couper");
            break;
        case "Copier":
            System.out.println("Copier");
            break;
        case "Coller":
            System.out.println("Coller");
            break;
        }
    }

    public void itemStateChanged(ItemEvent e) {
        System.out.println(e.paramString());
    }

    public static void main(String args[]) {
        MenuRaccourcis f = new MenuRaccourcis("Avec raccourcis");
        f.setSize(380, 145);
        f.setVisible(true);
        f.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Remarques :

- `actionPerformed()` traitera les événements sur les trois premiers items du menus ;
- `itemStateChanged()` traitera les événements de l'entrée de menu de type `JCheckBoxMenuItem`.

Dans un terminal, à l'exécution :

```
[17:33][Prog pc666 :]$ java MenuRaccourcis
Copier
Coller
Couper
ITEM_STATE_CHANGED,item=javax.swing.JCheckBoxMenuItem[...
  ...],stateChange=SELECTED
```

# Les menus avec SWING – menu contextuel

Un menu contextuel est un menu de type `JPopUpMenu`[1] qui s'obtient à l'aide d'un clic droit de la souris.

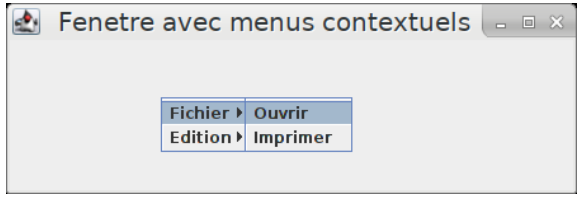
```
import javax.swing.*;
import static javax.swing.JFrame.*;
import java.awt.event.*;

class ExempleJPopupMenu extends MouseAdapter {
    JPopupMenu menuContextuel;
    JMenu fichier, edition;
    JMenuItem ouvrir, imprimer, couper, copier, coller;

    ExempleJPopupMenu() {
        // Création d'un menu contextuel/surgissant :
        menuContextuel = new JPopupMenu();
        // Création des menus :
        fichier = new JMenu("Fichier");
        edition = new JMenu("Edition");
        menuContextuel.add(fichier);
        menuContextuel.add(edition);
        // Création des entrées des menus :
        ...
    }

    public void mousePressed(MouseEvent e) {
        if (e.isPopupTrigger()) {
            menuContextuel.show(e.getComponent(),
                                e.getX(),
                                e.getY());
        }
    }                                                    ...
```

```
    public static void main(String args[]) {
        JFrame f = new JFrame("Fenetre avec menus contextuels");
        ExempleJPopupMenu m = new ExempleJPopupMenu();
        f.addMouseListener(m);
        f.setSize(700, 500);
        f.setVisible(true);
        f.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Remarque :

- cette méthode permet de déterminer s'il s'agissait d'un clic droit.

À l'exécution :




[1] https://docs.oracle.com/javase/8/docs/api/javax/swing/JPopupMenu.html

# Les listes avec JList<E> – 1/2

Exemple : un nuancier affichant les couleurs statiques de la classe `Color`.

Objectifs :

1. récupérer toutes les couleurs `static` de la classe `Color` ;
2. afficher leurs noms dans une liste `JList<E>` [1] ;
3. afficher une couleur sur simple clic dans la liste (grâce à l'interface *ListSelectionListener* [2]).

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.lang.reflect.Field;

class NuancierColor {
    public static void main(String args[]) throws IllegalAccessException {
        JFrame f = new JFrame("Nuancier de Color");
        InterieurFenetre intFen = new InterieurFenetre();
        f.setContentPane(intFen);
        f.pack();
        f.setVisible(true);
        f.setResizable(false);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
                                                                    ...
```

À l'exécution :


[1] https://docs.oracle.com/javase/8/docs/api/javax/swing/JList.html
[2] https://docs.oracle.com/javase/7/docs/api/javax/swing/event/ListSelectionListener.html

# Les listes avec JList – (2/2)

```
class InterieurFenetre
    extends JPanel
    implements ListSelectionListener {

    String tabNomsCouleurs[];
    Color tabCouleurs[];
    JList<String> JLCouleurs;
    JPanel zoneDessin = new JPanel();

    InterieurFenetre() throws IllegalAccessException {
        Field champsColor[] = (Color.class).getFields();
        int nbCouleurs = 0;
        for (Field c : champsColor) {
            if (c.getType() == Color.class) {
                nbCouleurs++;
            }
        }
        tabNomsCouleurs = new String[nbCouleurs];
        tabCouleurs = new Color[nbCouleurs];
        int i = 0;
        for (Field c : champsColor) {
            if (c.getType() == Color.class) {
                tabNomsCouleurs[i] = c.getName();
                tabCouleurs[i] = (Color) c.get(c);
                i++;
            }
        }
        JLCouleurs = new JList<String>(tabNomsCouleurs);
        JLCouleurs.addListSelectionListener(this);
        add(JLCouleurs);
        zoneDessin.setPreferredSize(new Dimension(450, 450));
        add(zoneDessin);
    }                                                    ...
```

```
    public void valueChanged(ListSelectionEvent e) {
        int numCouleurSelec = JLCouleurs.getSelectedIndex();
        zoneDessin.setBackground(tabCouleurs[numCouleurSelec]);
    }
}
```

## Remarques :

- Usage de la réflexivité[1] pour obtenir les seuls champs de type `Color` de la classe `Color`.
  - `getFields()` permettra d'obtenir tous ces champs ;
  - `get()` est susceptible de lever une exception (d'où les clauses `throws` de ce code).
- Création d'un objet de type `JList<String>` et ajout de son écouteur.
- Implémentation de la seule méthode de l'interface *`ListSelectionListener`*.
  - ici, on interroge directement l'objet de type `JList<String>` avec la méthode `getSelectedIndex()`.

[1] *cf.* fin du CM3.

# Ajouter des barres de défilement avec JScrollPane

Et si la liste des couleurs avait été très longue...

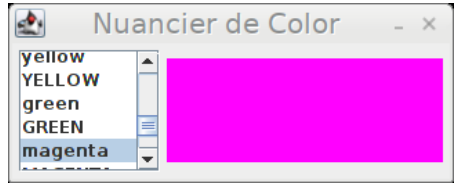
```
import java.util.Hashtable;
import java.util.Vector;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.lang.reflect.Field;

class InterieurFenetre
    extends JPanel
    implements ListSelectionListener {

    Hashtable<String, Color> HT = new Hashtable<String, Color>();
    JList<String> JLCouleurs;
    JPanel zoneDessin = new JPanel();

    InterieurFenetre() throws IllegalAccessException {
        Field champsColor[] = (Color.class).getFields();
        Vector<String> vecteurClefs = new Vector<String>();
        for (Field c : champsColor) {
            if (c.getType() == Color.class) {
                String nomCouleur = c.getName();
                vecteurClefs.add(nomCouleur);
                HT.put(nomCouleur, (Color) c.get(c));
            }
        }
        JLCouleurs = new JList<String>(vecteurClefs);
        JLCouleurs.addListSelectionListener(this);
        add(new JScrollPane(JLCouleurs));
        JLCouleurs.setVisibleRowCount(5);
        zoneDessin.setPreferredSize(new Dimension(200, 75));
        add(zoneDessin);
    }
    ...
```

```
    public void valueChanged(ListSelectionEvent e) {
        Color c = HT.get(JLCouleurs.getSelectedValue());
        zoneDessin.setBackground(c);
    }
}
```

Remarques :

- simplification du constructeur avec une table de hachage mettant en relation les noms de couleur avec les objets de type `Color` associés ;
- utilisation d'un objet de type `Vector<String>` pour faciliter la construction ;
- définition d'un objet de type `JScrollPane` qui ≪ contiendra ≫ notre objet de type `JList<String>`.

À l'exécution :



# Afficher une image

JAVA peut manipuler les formats d'image ayant ces extensions :

`bmp/wbmp`, `gif`, `jpg/jpeg`, `png`

En programmation, l'affichage d'une image :

- mobilise une ressource lente, c'est-à-dire l'écran ;
- peut bloquer momentanément le programme.

JAVA propose plusieurs techniques pour manipuler les images, notamment :

chargement avec attente

chargement sans attente

chargement avec gestion fine.

JAVA propose aussi des classes et des paquetages dédiés, dont :

| | |
|---|---|
| `IconImage` | pour les icônes - avec attente |
| `BufferedImage` | plus élaborées - plus lourd en mémoire |
| `VolatileImage` | idem - moins ≪ fiable ≫ que la version avec tampon |
| `ImageIO` | paquetage |

# Afficher une image – avec attente et ImageIcon

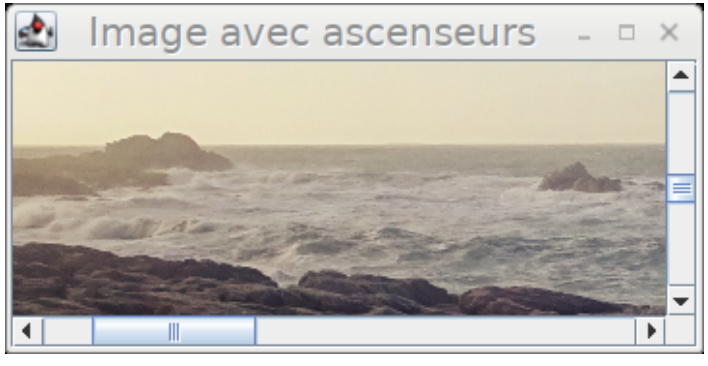
Exemple : affichage d'une image dans une fenêtre avec ascenseurs.

```
import java.awt.*;
import javax.swing.*;

class ImageAvecImageIcon extends JPanel {
    ImageIcon image;
    int largeurImage;
    int hauteurImage;

    ImageAvecImageIcon() {
        image = new ImageIcon("../Figure/coteBretonne.JPG");
        largeurImage = image.getIconWidth() / 4;
        hauteurImage = image.getIconHeight() / 4;
        this.setPreferredSize(new Dimension(largeurImage,
                                            hauteurImage));
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(image.getImage(), 0, 0,
                    largeurImage, hauteurImage, null);
    }

    public static void main(String args[]) {
        JFrame f = new JFrame("Image avec ascenseurs");
        ImageAvecImageIcon panneauImage = new ImageAvecImageIcon();
        JScrollPane ascenceurs = new JScrollPane(panneauImage);
        ascenceurs.setPreferredSize(new Dimension(350, 350));
        f.add(ascenceurs);
        f.pack();
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Remarques :

- création et affichage d'une image ;
- création d'ascenseurs pour la fenêtre.

À l'exécution :



# Afficher une image – sans attente et BufferedImage (1/2)

Exemple : affichage d'une image avec zoom *via* la roue de la souris.

```
... // imports classiques (i.e. java.awt.*, java.awt.event.*, javax.swing.*)
import java.io.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;

class ImageAvecZoom
    extends JPanel
    implements MouseWheelListener {
    static final float ZOOM_MIN = 1f;
    static final float ZOOM_MAX = 5f;
    static final float ZOOM_DELTA = 0.1f;
    static float ZOOM = 2.5f;
    BufferedImage image;
    int largeurImage, hauteurImage;

    ImageAvecZoom() {
        File fichierImage = new File("../Figure/coteBretonne.JPG");
        try {
            image = ImageIO.read(fichierImage);
        } catch (IOException e) {
            e.printStackTrace();
        }
        ajusterImage();
        this.setPreferredSize(new Dimension(image.getWidth() / 5,
                                            image.getHeight() / 5));
        addMouseWheelListener(this);
    }

    private void ajusterImage() {
        hauteurImage = (int) (image.getHeight() / ZOOM);
        largeurImage = (int) (image.getWidth() / ZOOM);
    }                                                                    ...
```

Remarques :

- Ajout de différents imports :
  - `java.io.*` : pour lire un fichier ;
  - `javax.imageio.ImageIO` pour lire et charger l'image en mémoire.
- Ouverture d'un fichier, suivie de la création de l'image en mémoire.

À l'exécution :

# Afficher une image – avec attente et BufferedImage (2/2)

```
    private void zoom(int changement) {
        if ( (changement > 0) && (ZOOM < ZOOM_MAX) ) {
            ZOOM += ZOOM_DELTA;
        }
        else if ( (changement < 0) && (ZOOM_MIN < ZOOM) ) {
            ZOOM -= ZOOM_DELTA;
        }
        ajusterImage();
    }

    public void mouseWheelMoved(MouseWheelEvent e) {
        zoom(e.getWheelRotation());
        repaint();
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(image, 0, 0,
                    largeurImage, hauteurImage, null);
    }

    public static void main(String args[]) {
        JFrame f = new JFrame("Image avec zoom");
        ImageAvecZoom panneauImage = new ImageAvecZoom();
        f.add(panneauImage);
        f.pack();
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Remarques :

- implémentation de l'interface *mouseWheelListener*
  - ajustement avec notre méthode `zoom()` en interrogeant l'objet `e` de type `MouseWheelEvent`.
  - invocation de la méthode `repaint()`.

# Afficher une image – sans attente avec Toolkit/MediaTracker (1/2)

Exemple : affichage d'une image avec zoom réglable avec une règle.

```
import java.util.Hashtable;
import java.awt.*;
import javax.swing.*;
import static javax.swing.JFrame.*;
import javax.swing.border.*;
import javax.swing.event.*;

class ImageJSlider extends JPanel {
    static final int ZOOM_MIN = 0;
    static final int ZOOM_MAX = 50;
    ImageADessiner image;

    ImageJSlider(String chemin) {
        setLayout(new BoxLayout(this, BoxLayout.PAGE_AXIS));
        image = new ImageADessiner(chemin);
        add(image);
        //Création d'un label et d'une règle.
        JLabel zoomeurLabel = new JLabel("Zoom", JLabel.CENTER);
        zoomeurLabel.setAlignmentX(Component.CENTER_ALIGNMENT);
        JSlider zoomeur = new JSlider(JSlider.HORIZONTAL,
                                      ZOOM_MIN, ZOOM_MAX, ZOOM_MIN);
        zoomeur.addChangeListener(image);
        // Redéfinition des marques écrites de la règle (int <-> float).
        Hashtable<Integer, JLabel> HT = new Hashtable<Integer, JLabel>();
        for (int i = ZOOM_MIN; i <= ZOOM_MAX; i += 10) {
            HT.put(new Integer(i), new JLabel("" + i / 10.0f));
        }
        zoomeur.setLabelTable(HT);
        zoomeur.setMajorTickSpacing(10);
        zoomeur.setMinorTickSpacing(5);
        zoomeur.setPaintTicks(true);
        zoomeur.setPaintLabels(true);
        Border bordure = BorderFactory.createEmptyBorder(10, 10, 10, 10);
                                                                    ...
```

```
        zoomeur.setBorder(bordure);
        zoomeurLabel.setBorder(bordure);
        add(zoomeurLabel);
        add(zoomeur);
    }

    public static void main(String args[]) {
        JFrame f = new JFrame("Image avec JSlider");
        f.setDefaultCloseOperation(EXIT_ON_CLOSE);
        ImageJSlider imgJS = new ImageJSlider(args[0]);
        f.add(imgJS);
        f.pack();
        f.setResizable(false);
        f.setVisible(true);
    }
}
```

À l'exécution :

# Afficher une image – sans attente avec Toolkit/MediaTracker (2/2)

```
class ImageADessiner
    extends JPanel
    implements ChangeListener {

    Image image;
    MediaTracker pisteur;
    float zoom = 0.5f;
    int largeurImage, hauteurImage;
    int largeurMax, hauteurMax, largeur, hauteur;

    ImageADessiner(String chemin) {
        image = getToolkit().createImage(chemin);
        pisteur = new MediaTracker(this);
        pisteur.addImage(image, 0);
        try {
            pisteur.waitForID(0);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        if (pisteur.statusID(0, false) != MediaTracker.COMPLETE) {
            System.out.println("Erreur de chargement");
            return;
        }
        largeurImage = image.getWidth(this);
        hauteurImage = image.getHeight(this);
        largeurMax = largeurImage / 8;
        hauteurMax = hauteurImage / 8;
        largeur = (int) (largeurMax / 2.5);
        hauteur = (int) (hauteurMax / 2.5);
        setPreferredSize(new Dimension(largeurMax,
                                       hauteurMax));
    }
```

...

## Remarques :

- chargement de l'image avec un objet de type `MediaTracker`[1] ;
- assurance que l'image est bien disponible ;
- traitement des événements de l'objet de type `JSlider`.

```
    public void stateChanged(ChangeEvent e) {
        int valJS = (((JSlider) e.getSource()).getValue());
        zoom = 0.5f * (1 - valJS * (1.0f / 50));
        repaint();
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(image,
                    0, 0, largeurMax, hauteurMax,
                    (int) ((0.5 - zoom) * largeurImage),
                    (int) ((0.5 - zoom) * hauteurImage),
                    (int) ((0.5 + zoom) * largeurImage),
                    (int) ((0.5 + zoom) * hauteurImage),
                    this);
    }
}
```

[1] https://docs.oracle.com/javase/8/docs/api/java/awt/MediaTracker.html

# Animation en JAVA – principe

Cela se fait à l'aide d'un objet de type `Timer`[1] qui générera des événements de type `ActionEvent`.

Pour créer un tel objet :

```
Timer t = new Timer(delai, objEcouteur);
```

Principales méthodes associées :

```
// Enclencher/stopper le Timer :
void start();
void restart();
void stop();
// Jouer sur le délai :
int getDelay();
void setDelay(int);
void setInitialDelay(int);
// Caractère répétitif du Timer :
boolean isRepeats();
void setRepeats(boolean);
```

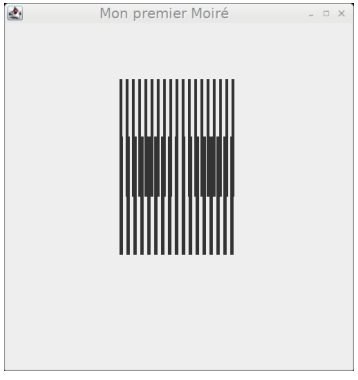
Le programmeur implémentera alors l'interface *ActionListener* afin de traiter périodiquement ces signaux.

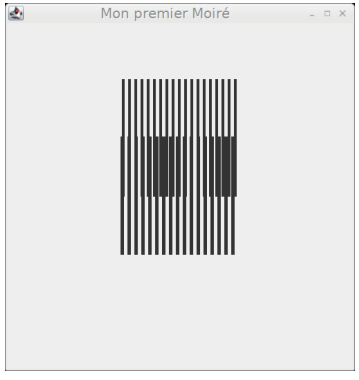
[1] https://docs.oracle.com/javase/8/docs/api/javax/swing/Timer.html

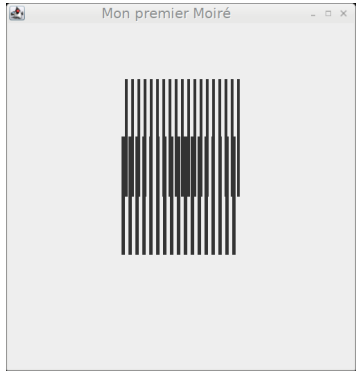
# Animation en JAVA - exemple 1/2

Un ≪ moiré ≫ [1] est un effet visuel reposant sur la création d'interférences grâce à la superposition de différents motifs.

Utilisé dans une animation, ce motif d'interférences semble se déplacer.








*Grosso modo* pour réaliser un tel moiré, on dispose deux grilles telles que :

- l'épaisseur de chaque barreau de la grille centrale soit à peine supérieure à celle de chaque barreau de la grille mobile ;
- pour une certaine longueur $\mathcal{L}$, le nombre de barreaux de la grille mobile excède de un le nombre de barreaux de la grille centrale.

Il suffira alors de décaler périodiquement vers la droite la grille mobile.

[1] https://en.wikipedia.org/wiki/Moir%C3%A9_pattern

# Animation en JAVA – exemple 2/2

```
import java.awt.*;
import javax.swing.*;
import static javax.swing.JFrame.*;
import java.awt.event.*;

class Moire
    extends JPanel
    implements ActionListener {

    int x;
    int y;
    // Épaisseurs des barreaux :
    float epGrMobile, epGrCtrale;
    int decalage;
    Timer metronome;

    Moire(int x, int y, float epGrMobile) {
        this.x = x / 3;
        this.y = y / 3;
        this.epGrMobile = epGrMobile;
        epGrCtrale = epGrMobile + 0.5f;
        decalage = 0;
        setPreferredSize(new Dimension(x, y));
        metronome = new Timer(80, this);
        metronome.start();
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == metronome) {
            repaint();
            decalage = (decalage - 1) % (8);
        }
    }
```

...

```
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        // Dessin de la grille centrale :
        for (int i = x; i < 2 * x; i += ((int) (2 * epGrCtrale))) {
            ((Graphics2D) g).setStroke(new BasicStroke(epGrCtrale));
            g.drawLine(i, y, i, 2 * y);
        }

        // // Dessin de la grille mobile :
        for (int i = x; i < 2 * x; i += ((int) (2 * epGrMobile))) {
            ((Graphics2D) g).setStroke(new BasicStroke(epGrMobile));
            g.drawLine(i + decalage, x / 2, i + decalage, x / 2 + x);
        }
        g.dispose();
    }

    public static void main(String args[]) {
        Moire m = new Moire(500, 500, 4.5f);
        JFrame f = new JFrame("Mon premier Moiré");
        f.setContentPane(m);
        f.pack();
        f.setVisible(true);
        f.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Le mouvement de la grille mobile est réalisé en la redessinant périodiquement avec un décalage.

# Jouer un son avec JAVA

La JVM autorise de manipuler des fichiers audios [1] (voire vidéos) notamment grâce au paquetage `javax.sound.sampled` [2].

Exemple : jouer un meuglement [3].

```
import java.io.File;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.Clip;
import javax.sound.sampled.DataLine;

class LecteurAudio {
    static void jouerSon(File fAudio) {
        try {
            AudioInputStream ais = AudioSystem.getAudioInputStream(fAudio);
            DataLine.Info info = new DataLine.Info(Clip.class, ais.getFormat());
            Clip c = (Clip) AudioSystem.getLine(info);
            c.open(ais);
            c.start();
        } catch (Exception e) {
            System.out.println("Erreur au niveau du son : " + e);
        }
    }

    public static void main(String args[]) {
        File fs = new File("/usr/lib/libreoffice/share/gallery/sounds/cow.wav");
        jouerSon(fs);
    }
}
```

Remarques :

- ces lignes permettent d'ouvrir et de jouer le son ;
- cela permet d'ouvrir un fichier donné (les E/S ne seront pas traitées dans ce cours).

[1] https://www.oracle.com/technetwork/java/javase/formats-138492.html

[2] https://docs.oracle.com/javase/7/docs/api/javax/sound/sampled/package-frame.html

[3] Trouvé à l'aide de la commande "`locate -i *.wav`".